Programming 2
Tutorial and Practical 11

This week's tutorial and practical sessions are about self-referential node structures, and searching and sorting algorithms..

Tutorial Session

1. Outline the limitations linked lists pose on being able to traverse and manipulate data items within the structure.

2. Consider a linked list that stores a series of integers, and arranges data items such that they are *always stored in sort order*. Trace every traversal, node and reference manipulation in performing the following functions (assume an empty list to begin with):

- **a.** Add a data item "17".
- **b.** Add a data item "142".
- **c.** Add a data item "46".
- **d.** Search for the data item "92".
- **e.** Add a data item "199".
- **f.** Delete the data item "142".
- **g.** Delete the data item "17".

At each step, consider why it is important for the node and reference manipulations to be done in a specific order. What would be the consequence of doing it differently?

3. The linked lists introduced in the course notes are very simple, in that they tend to store single items of data. How can a linked list be changed so that it can perform *useful* storage work? (i.e. carry one, or a series of, objects)?

- **a.** What parts of the list definition need to be changed?
- **b.** How will the list be able to identify a particular piece of data?
- **c.** How should data be returned from the list?

4. Could a binary search be implemented on a linked list? Would it be a useful thing to do? Why (or why not)?

5. Why must data be sorted before a binary search can take place? What sort of assumptions does the algorithm make on the way data is arranged?

6. While tracing your steps, arrange the following set of data by a selection sort:

```
12 83 72 84 27 11 93 72 46
```

Once complete, perform (while tracing your steps) a binary search for 12 and 71.

Practical Session

Consider the following basic class for storing a telephone number:

```
class PhoneNumber
{
   String name;
   int number;
}
```

Implement a linked list that stores `PhoneNumber` objects. Phone numbers are to be uniquely identified by the name, and as such duplicate names should never be stored. The following functionality should be present:

- *Add data:* given a `PhoneNumber`, it should store it (after checking that there isn't already a matching number). Data should be kept in sort order.
- *Retrieve data:* given a name `String`, it should retrieve the `PhoneNumber` object.
- *Delete data:* given a name `String`, it should delete the appropriate `PhoneNumber` object, if it exists.
- *Summarise data:* create an array of `PhoneNumber` objects based on the data in the list, and return it.

Write a very simple driver class to test each function of the list structure. Data validation is not required for the driver class, however your linked list must fail gracefully (for example, in cases where an attempt is made to add data with the same name; or retrieve or delete data that doesn't exist.)